Use of Agents to Implement an Integrated Computing Environment

Mark A. Hale* and James I. Craig†

Aerospace Systems Design Laboratory
Georgia Institute of Technology
School of Aerospace Engineering
Atlanta, Georgia 30332-0140

http://www.cad.gatech.edu/image

Abstract

Integrated Product and Process Development (IPPD) embodies the simultaneous application of both system and quality engineering methods throughout an iterative design process. The use of IPPD results in the time-conscious, cost-saving development of engineering systems. To implement IPPD, a Decision-Based Design perspective is encapsulated in an approach that focuses on the role of the human designer in product development. The approach has two parts and is outlined in this paper. First, an architecture, called DREAMS, is being developed that facilitates design from a decision-based perspective. Second, a supporting computing infrastructure, called IMAGE, is being designed. Agents are used to implement the overall infrastructure on the computer. Successful agent utilization requires that they be made of three components: the resource, the model, and the wrap. Current work is focused on the development of generalized agent schemes and associated demonstration projects. When in place, the technology independent computing infrastructure will aid the designer in systematically generating knowledge used to facilitate decision-making.

Background

Considerable time and effort has been invested in the development of new computing technologies and their associated methods for **I**ntegrated **P**roduct and **P**rocess **D**evelopment (IPPD). Unfortunately, these efforts have resulted in implementations that are disjoint in their application. A Decision-Based Design approach is taken that focuses these efforts on the development of a structured decision support process that originates from the designer's perspective. A formal, structured technique for the embodiment of a decision-based design perspective and a corresponding computer-based implementation scheme is taken in the approach.

* Graduate Researcher, Student Member AIAA

† Professor, Member AIAA

AIAA-95-1001
AIAA - Computing in Aerospace 10
San Antonio, TX, March 28-30, 1995

Computer-based resources play a significant role in generating knowledge about a design. Considerable time and expense has been given to the development of the computing technologies required for better resource efficacy. These technologies have been applied in systems that emphasize modularity, interdisciplinary program utilization, resource collaboration, and distributed processing.[1-8] These systems have marked improvements in information processing. However, their applicability to aiding the designer in making decisions based on new design knowledge remains questionable. Furthermore, the applicability of these systems to a continuous, iterative design life-cycle has not been shown and is uncertain.

A two-step approach is being taken in the design of a new computing infrastructure for assisting a designer in making decisions. First, a coherent, systematic decision-making architecture that is used to structure, but not restrict, the means by which the designer solves the design problem (story) is formalized. Second, after the necessary components of the new technique are identified, an open computing infrastructure is designed to explicitly include support for these design related components. Then, specific tools for successful computing operation are identified independently of the design related activities. An enabling technology for the infrastructure is the agent and is the subject of this paper.

The approach used here will result in a computing environment that will serve to implement IPPD. The resulting implementation will:

- *Facilitate designing from a decision-based perspective;*
- *Provide a means for both structuring the design problem and of solving the problem; and*
- *Focus the application of new and existing computing technologies toward assisting the designer in applying engineering methods.*

The two elements of this approach are outlined in this paper. First, an architecture that facilitates design from a decision-based perspective is formalized. Secondly, a suitable computing infrastructure used to implement the design architecture is presented. As an enabling technology, agents will be formally defined and a generic agentization scheme proposed.

A Designer Makes Decisions

The basic premise set forth in the design of the computing infrastructure is that the framework exists to aid the designer in making decisions. Fundamentally, the principal role of the designer is to make decisions throughout the design process. A paradigm for capturing this perspective is ***D**ecision **B**ased*

***D**esign* (DBD). One embodiment of DBD is the ***D**ecision **S**upport **P**roblem **T**echnique* (DSP Technique).[9-11] The DSP Technique is used to implement IPPD from a Decision-Based Design perspective. Integrated Product and Process Development embodies the simultaneous application of system and quality engineering methods throughout the iterative design process. Other implementations focus on the identification and application of methods to be employed in IPPD. The DSP Technique builds on these implementations and facilitates IPPD by providing a designer a means for structuring design processes into an organized solution scheme, utilizing Support Problems. Not only is the design problem defined, but an achievable solution process is known.

DREAMS - An Architecture that Facilitates Decision-Based Design

The architecture that is being developed aids a designer in solving a design problem and is shown in Figure 1. Based on the DSP Technique described in the previous section, the architecture has three fundamental components:

- *A designer's perspective*, the role of the designer is to make decisions throughout a design's life-cycle;
- *Support Problem definition and solution*, knowledge about the design is used by the designer to make decisions and Support Problems model the transformation of information into knowledge; and
- *Design management*, knowledge about the design is appropriately managed so that it can be used in making decisions.

The implementation of this architecture will aid the designer in **D**eveloping **R**obust **E**ngineering **A**nalysis **M**odels and **S**pecifications. Thus, the architecture is named DREAMS.

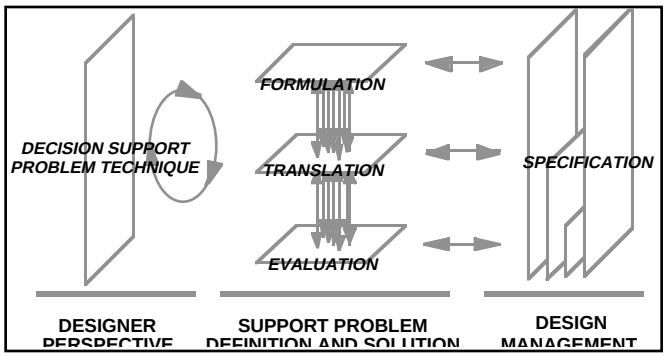

Figure 1. DREAMS Architecture

A Designer's Perspective

There are two phases in the DSP Technique:

- *The meta-design phase*, whereby the designer designs the design process with the aid of Support Problems; and
- *The actual design phase*, whereby Support Problems are exercised so that knowledge about the design can be generated and decisions can be made.

A meta-design phase allows a designer to explicitly model design process using Support Problems. After laying out the design process in meta-design, a designer can use the DSP Technique to generate knowledge used for decision-making through the use of Support Problems. Support Problems provide a model that governs the creation of new design information. One kind of Support Problem is the Decision Support Problem and is used to explicitly declare a decision making process. A sub-class of DSP's is the Compromise DSP. The Compromise DSP has a linguistic statement comprising of the *Given*, *Satisfy*, *Find*, *Minimize* keywords.

Support Problem Definition and Solution

Within the DSP Technique, Support Problems are exercised by a designer to produce knowledge about a design so that decisions can be made based on that knowledge. Support Problems provide standard models for transforming design information into knowledge. There are three steps required in defining and solving Support Problems:

- *Formulation*, the structuring of the problem statement into specific Support Problem models;
- *Translation*, associating processes, that govern the generation of information into knowledge, with the Support Problems; and
- *Evaluation*, producing design knowledge through the solution of the Support Problems.

Formulation - Support Problem

Support Problems are defined when the design process is partitioned in meta-design. Support Problems have a defined structure given by keywords. The Compromise DSP has the form:[12]

Given:	Feasible Design and Aspiration Space
Find:	Values of Variables
Satisfy:	Systems Constraints, Bounds, and Goals
Minimize:	Deviation between "what I want" and "what I can have"

Support Problems are formulated as linguistic statements, a form natural to the designer and, hopefully, unambiguous in meaning.

Translation - Math Form

Once a Support Problem has been formulated, the problem is translated into an equivalent Math Form. The Math Form provides the process connectivity between forms and functions. For instance, the functions `lift` and `drag` are associated with the form `wing` through the relation:

$$\frac{L}{D} = \frac{C_l}{C_{D_o} + K C_l^2} \qquad (1)$$

As the Math Form becomes more complex, equations are typically grouped into engineering models. In turn, models are often grouped into disciplines.

Evaluation - Template

Finally, the Math Form of the Support Problem can be solved. The Support Problem solution consists of three steps: pairing the Math Form with a suitable Design Operator, structuring a solution network, and solving the Problem. A Design Operator generates additional design information from the expressions found in the Math Form of a Support Problem. (The familiar computational counterpart to the Design Operator is the *Agent*.) Design Operators are typically

engineering analysis codes. Other Design Operators include expert systems, hyper-media sources, virtual reality, and the human designer. The combination of the Math Form and the Design Operator is called a Design Event. The Design Event is important because during its instantiation knowledge about the design is generated. After the Math Form and Design Operators have been collected, the new form of the Support Problem is called the Support Problem Template.

Continuing, a solution network must be generated for the Design Events. Finally, the Support Problem must be solved. **D**ecision **S**upport **I**n the **D**esign of **E**ngineering **S**ystems (DSIDES) is a suite of tools used to solve Support Problems.[13] Tools in DSIDES can be used to solve Selection DSPs (SELECT) and multi-level, multi-goal Compromise DSPs (ALP).

Design Management

Design management aids a designer in reviewing knowledge for decision-making and is included in the DREAMS architecture, see Figure 1. A suitable design management scheme is extremely important since information is used in decision-making throughout the life-cycle of design. The knowledge gained about the design must be accurate, accountable, and time-consistent. Further complicating matters, the amount of design information produced in a design is extremely large, and widely distributed. Successful design management requires three fundamental components:

- *The structure of information*, the means by which information is organized as a design progresses;
- *The measurement of information*, the ability to quantify the progression of a design; and
- *Information access*, large, distributed storage and retrieval schemes.

IMAGE - A Computing Infrastructure

Having captured a designer's perspective in an architecture that supports Decision-Based Design, a computing infrastructure is being developed that provides a coherent implementation of the architecture. The computing infrastructure is designed in two parts:

- *Explicit entities are used to directly implement the DREAMS architecture; and*
- *An environment combines these entities with supporting computational tools.*

The resulting infrastructure is called IMAGE, an **I**ntelligent **M**ultidisciplinary **A**ircraft **G**eneration **E**nvironment. The infrastructure began as a special project that recognized the lack of designer support in traditional frameworks.[14] This fact resulted in frameworks that are difficult to implement and has lead to their limited use.

IMAGE is a loosely configured, agent/tool-based federation. An agent environment supports multiple platforms, operating systems, and users. The resulting IMAGE infrastructure is shown in Figure 2. Tools can be used in place of agents when a design expert is present to utilize the resources and are required for operations that have no model (for example operating system level services). The following section will formalize agents specifications and will present a generalized agent scheme for utilizing resources in a integrated computing infrastructure.

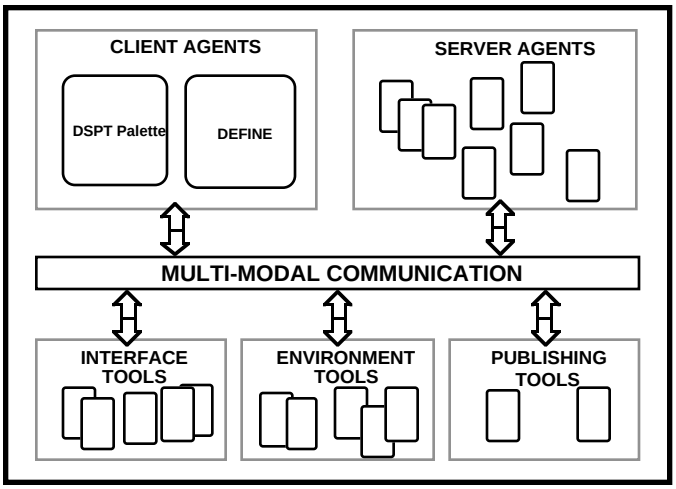

Figure 2. IMAGE Infrastructure

Agents

There are two types of agents that will exist in the environment: the client and the server. Client agents are those agents that provide user services for directing design. These agents are shown in Table 1. Server agents are the agents used to implement design resources. Table 2 summarizes a few of the server agents that can be used for aircraft design. Specific agents will be developed as example problems become better defined.

Table 1. Client Agents

Agent	Model
DSPT Palette	Meta-Design Support Problem definition and solution
DEFINE	Design specification

Table 2. Server Agents

Agent	Model
Geometry	Solids geometry construction
Convergence	Weight convergence, fuel balance
Aerodynamics	Newton-Euler, Navier-Stokes
Structures	Energy methods
Propulsion	Thermodynamics
Weight and Balance	Mass properties, Newton's Second Law
Controls	Control laws
Costing	Historical CERs

Tools

A suite of tools that can be used to assist in the design process and general environment services. These tools have been categorized as interface (both human and inter-agent), monitoring, and publishing tools. These tools transparently coordinate inter-agent, agent-tool, and inter-tool efforts. The services of these tools are summarized in the following tables:

Table 3. Human and Inter-Agent Interface Tools

Tool	Service
Database	Relational and object-oriented data management.
File	File management
Communications	Data routing through unsynchronized, unlike channels
Application Defaults	User-defined (graphical) user interface defaults
Security	User and process security
Recovery	Environment recovery
Lookup	Agent/tool name server
Dictionary	Cross-agent ontology correlation
Accumulator	Quantify design knowledge
Annotator	Record design events
Review	Review design decisions and history
Units	Agent independent standard units and conversions
Expert System	Distributed, rule-based inferencing capabilities
Heterarchy Editor	Loose information management
Language Processor	Natural language processing services. May grow to include pictures, movies, and sound.

Table 4. Environment Monitors

Tool	Service
Software	Software versioning and synchronization
Process	Resource execution status
Data	Data storage/retrieval utilities
User	User and process management
Project	Distributed project capabilities

Table 5. Knowledge Publishing

Tool	Service
Printing	Local and remote spooling
WWW	Electronic documentation

The IMAGE infrastructure provides the computational support for design decision-making. Specific, design-related tools were developed for the computing environment. The use of these tools provides a systematic mechanism for generating knowledge used for decision-making. Additional tools provide the resources required for collaboration in open, multi-user computing systems.

Agents and tools prove to be one of the key components required to implement the IMAGE infrastructure. The next section formalizes the structure required for agents so that they can be used to perform design-related activities.

Agents - An Enabling Technology

Agents provide a mechanism for resource use in a distributed, heterogeneous computing environment. The background for agents will be discussed leading to a formal agent definition. From this definition, the basic components of the agent will be isolated. Furthermore, the role of the tool, a specialized agent, in a computing environment will be established.

Definition

In their initial form, agents were simply considered to be programs that could communicate with each other, as given by the following definition.[2]

> "[An agent is] a computer program that communicates with external programs exclusively via a pre-defined protocol."

Efforts were and are currently focused on the development of communication standards and protocols. It is hoped that a standardized implementation mechanism will facilitate the use of agents. However, the use of agents in this form does not guarantee that agents can be used to successfully implement a design-oriented, integrated computing environment. It remains to be shown that the agent, as defined here, can be used to instantiate designer desires while conforming to a design specification. In this agent definition, two of the three necessary agent components are recognized: the resource, and the wrap. Simply stated, resources are computer programs and a wrap is the portion of the computer program used for program communications.

LEGEND research expanded on agents in their present form by adding a new component, the model.[15] The model is part of the finite-dimensional design space given by a design specification. Agents are defined in LEGEND to be:

> An agent is a resource that has been modeled and wrapped for inclusion in a distributed design environment. The environment dictates requirements for context-oriented documentation and publication and experience mechanisms.

Three components are now formally declared to be part of the agent: the resource, the model, and the wrap. The model provides a scheme for generating additional design information. This definition characterizes an agents external behavior in terms of its documentation and publication mechanisms, but fails to signify the computational requirements that are required for successful agent integration.

The following agent definition is proposed:

> An agent is a resource that has been modeled and wrapped for inclusion in a distributed design environment. Agent design requires a designer-centered, bi-directional wrap that is independent of proprietary boundaries and capable of supporting increasing fidelity models.

This definition characterizes an agent by its components and behavior. There are three agent components: the resource, the model, and the wrap. Agents must accommodate information obtained from heterogeneous resources and must apply this to design models of increasing fidelity across a product's life-cycle. Agents are one of the key integration tools for a distributed, designer-centered, multi-tasked design environment. It will also be shown later in this paper that agents are able to generate design information while maintaining accountability for all actions.

For the first time the role of proprietary resources and information is explicitly stated in the agent definition. Proprietary resources are generally stand-alone in nature, with

limited communications capabilities, and preserve software rights through a number of advanced computing techniques. Together, they present a formidable challenge to implementation of integrated design environments. Finally, it should be noted that, proprietary information must be accommodated and secured in open, integrated environments.

Agent Components

As shown in Figure 3, there are three agent components:

- *Resource*. Entities responsible for transforming information into knowledge. The computer program is the most commonly used design resource.
- *Model*. Models generalize the process by which the resource generates new knowledge about a design.
- *Wrap*. The wrap provides inter-agent communication capabilities and supports internal information processing.

All three components must exist in an agent's definition. Later, it will be shown that this requirement can be relaxed when a special agent, called a tool, is used. The model is the most design-oriented component and, inherently, is the most abstract. For this reason, the model will be discussed last. However, the model should be the first component to be defined in an agent.

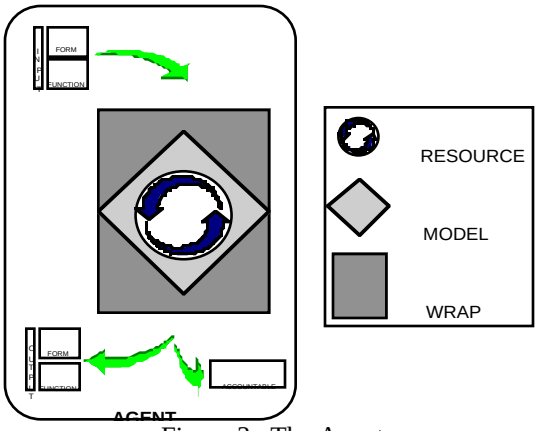

Figure 3. The Agent

Resource

Resources are the entities that are inevitably responsible for carrying out design methods. The most familiar form of a resource is the computer program. The computer program calculates some desired information based on pre-determined algorithmic procedures. Some examples of computer programs used by the aerospace industry include: ASTROS (a structural optimization code), FLOPS (an aircraft convergence code), ACSYNT (an aircraft convergence code), CONMIN (an optimization package), CATIA™ (a three-dimensional geometric modeling, simulation and analysis package), and ORACLE™ (a relational database). Figure 4 shows an example of a C program that, when given a number of points, calculates the coordinate-pairs representing the points on a unit circle.

Traditional systems that employ agents primarily operate in the conceptual stage of design. The design resources used in these systems are mostly non-proprietary codes, as illustrated in Figure 5. However, computing environments must also incorporate proprietary resources that predominate

```
/*
 *This program determines the points
 *profile for a unit circle.  N is
 *given on the command line.
 */
#include <stdio.h>
#include <math.h>
#define PI 3.141593
main (int argc,char *argv[])
{
 int N=atoi(argv[1]);
 int n;
 for ( n = 1; n <= N; n++)
    printf(" { %f %f } ",
       cos( (n-1)*2*PI/N),
       sin( (n-1)*2*PI/N));
}
```

Figure 4. Profile Program

later in a product's design life-cycle, as also seen in Figure 5. As discussed earlier, the integration of these kinds of resources is more difficult than the that of nonproprietary resources. A generic agentization scheme will need to incorporate new technologies to accommodate proprietary resources. These techniques will assure that environment frameworks are scalable when the frameworks are put into use. In addition, the ability to use proprietary resources presents a considerable step toward design automation.[14]

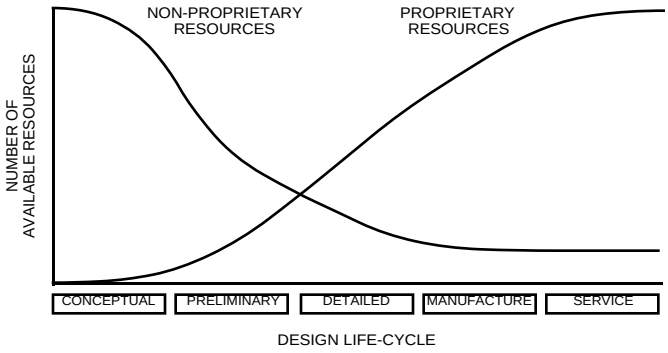

Figure 5. Proprietary Resources

Associated with the use of proprietary resources is information security. Not only do resources need to be protected, but information transfer/communication between them also has to be made secure. Techniques must be in place to guarantee secure, yet open, integrated environments.

There are a number of other resources that are often overlooked. The first is the design expert. The designer plays an important role in providing expert knowledge, which may be captured in knowledge-based systems. The importance of the designer as a resource is also evident in the fact that agents are "designer-centered" (from the definition of an agent). Another type of resource that is often overlooked is the design experience resource. Experience-related resources capture the "lessons learned" during design. Lessons learned may only be domain knowledge, but they may also consist of published reports, such as NASA TM reports, or on-line electronic

documentation, such as World Wide Web (WWW) documents.

Wrap

The wrap serves a dual purpose for agents. First, the wrap provides comprehensive information management. Second, proper wrap definition allows for standardized agent implementation. Most of the wrap is agent independent and is configured at run-time. Therefore, the agent can adapt to the current operating environment and user. The wrap has six components: the Communications Interface, the Protocol Filter, the Model Interpreter, the Resource Interpreter, the (Graphical) User Interface, and the Low Level Compliance layer.

The wrap utilizes a *Communications Interface* to facilitate bi-directional data exchange within the design environment. For computer-based resources, the communication channel is accessible through the multi-user, multi-platform, multi-language, networked workstation systems used in current design systems. Communications operate synchronously and asynchronously. A wrap includes an event loop that services standard communications traffic: including stdio, X-events, multi-domain socket interfaces, virtual file transfer, threads, hyper-text transfer protocol, and e-mail. The Communications Interface monitors events from the (Graphical) User Interface. The Communications Interface is agent independent and may be configured at run-time.

The flexible use of agents cannot be guaranteed unless the agent adheres to a protocol standard. A *Protocol Filter* provides a transfer format and a method for information encapsulation, the agent ontology. One transfer format being developed is the Knowledge Query Manipulation Language (KQML).[16] KQML provides the tools necessary to route information in an agent-based operating environment. The Protocol Filter also converts internal agent data into ontology compliant data and vice versa.[17] The correlation between the ontology and the model interpreter is still being investigated and further discussion can be found in the next section. The Protocol Filter is resource independent and should be de-coupled from other agent operations so that updates can be made in accordance with new standards. The transfer format is agent independent; however, the agent's ontology is agent specific.

Once the necessary information has been received and converted, the wrap contains mechanisms for processing it. Three facilities provided by the wrap: a *Model Interpreter*, a *Resource Interpreter*, and a *(Graphical) User Interface*. The Model Interpreter is a resource and model independent mechanism for interpreting models. The Model Interpreter will be discussed in more detail in the next section.

The Resource Interpreter presents a suite of integration tools for a variety or resources. This interpreter insures that the agent is compatible with existing resources by adapting to the resource's current implementation. Therefore, *resources do not have to modified to use them*. As mentioned earlier, the accessibility of design resources varies significantly between proprietary and non-proprietary codes. Nonproprietary codes are often easier to wrap because source code level access is available. Therefore, wrapping utilities can be directly integrated by restructuring the source code itself. In contrast, proprietary resources are usually provided in an object/executable form. Fortunately, internal resources of mature commercial software products can often be integrated with link-edit procedures, and this can form the basis for agent wrapping. In addition, backward compatibility is insured. Later, resources will be classified and interpreter schemes proposed. The Model Interpreter and the Resource Interpreter are tightly coupled for automated information exchange and both are agent independent.

The (**G**raphical) **U**ser **I**nterface (GUI) provides a standardized interface for user interaction, error handling, and process monitoring. Under X11 and an appropriate configuration, the interface may be graphical, otherwise the interface will use tty-based interactions. An interface will be provided that is familiar to the user across applications. Many applications due not have GUIs and operate in a variety of ways, from directed output to multiple-screen GUIs. The Graphical User Interface will provide a standard tool suite similar to the Macintosh. The User Interface is the only agent component that is optional. Without the interface, the agent must operate autonomously or through a non-standard IO channel.

The final component of the wrap is ***L****ow-****L****evel* ***C****ompliance* (LLC). Low-Level Compliance is the environment and agent specific tasks entailed during agent use. Such tasks include agent registration and validation, process monitoring, run-time publication, security checks, and dictionary/thesaurus lookups. LLC is configured at run-time and may be dynamically altered throughout an agent's life. LLC operations are processed by the wraps' event loop but are transparent to the model and resource interpreters. The LLC may be used by the Graphical User Interface, as in the case of process monitoring. LLC also provides a location for local data storage and retrieval. LLC provides the services of "facilitators" found in other agent-based systems.[2]

Model

The model has two components: the *Process Model* and the *Implementation Model*. The Process Model comes from the design specification, as given by LEGEND.[15] The process is the manner by which a form achieves its function. Some processes may be physical, an `aerodynamic` process; or it may intellectual, a `decision` is made. These processes are modeled so that engineering analyses can be made. A `2-D vortex panel` may be the model used to determine aerodynamic forces for an `aerodynamic` process. The `Decision-Based Design` paradigm may be modeled using the `Decision Support Problem Technique.` Other models are typically based on mathematical formulations, engineering principles, or geometrical constructions.

The Process Model has typically been discarded or included in an external publication. The agent allows for the Process Model to be explicitly defined. Figure 6 shows a solid construction used to represent complex solids in CATIA. In words, the geometric process model describing the volume transformation would be:

> In a volume transformation, an object is represented by an approximate solid computed in CATIA directly from the exact volume. A volume is constructed from

faces which, in turn, are defined by the edges that enclose simple or multiply connected regions of planar or complex surfaces.

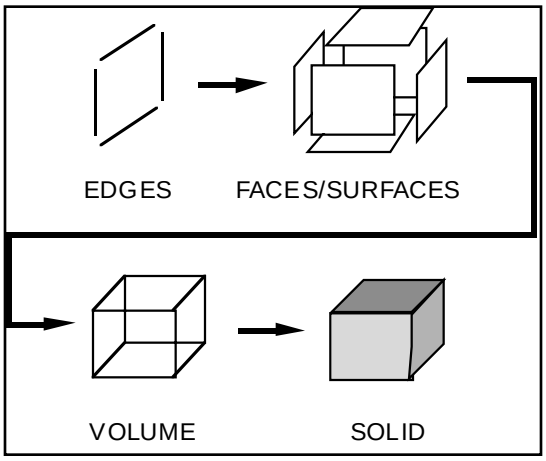

Figure 6. CATIA Solid Representation

A mechanism for describing models needs to be developed. The Knowledge Interchange Format (KIF) is a way for describing models (or ontologies).[18] Models described in KIF start with basic principles and then are combined to form higher-order methods. The description used for KIF is done using LISP-like constructs. Another candidate for model description would be to use a form of natural language processing similar to that used to formulate support problems.[12] (NLP in this context also includes pictures, sound, movies, etc.) For instance, the volume transformation linguistically described above would be a valid model.

Process models are usually coded for automated analysis. Given some input form and function information, the programs determine new form and function information. These computer programs are the resources of agents, see resource discussion above.

The Implementation Model, the second model component, captures the execution characteristics of the resource. Some of the items that are contained in the implementation model include:

- *Definition*. Form and function schemas are locally declared. For example a `lifting surface` form may be described with `cord` and `span` in one program and `aspect ratio` and `span` in another.
- *Units*. Representation units are explicitly declared. For example, a resource may require that `span` be in `feet`. Or, it may be that the resource is non-dimensional. In that case, the non-dimensional parameter is given. For example, the `span` has unit `length`.
- *Execution*. The means by which the program executes is defined. Some programs require standard input channels, e.g. "`run_me < file.in`". While others require specific files, such as FLOPS, which requires an input file named `FLOPS.IN`. In addition, data files are declared so that they can be managed during recursive resource calls.
- *Platform*. The platforms that the program can run on and has been compiled are declared. This information facilitates execution in a distributed, heterogeneous environment.

Similar to the process model, a method for describing implementation models will need to be developed.

Referring to the previous section, the *Model Interpreter* found in the wrap is used to exercise the process and implementation models. In addition, the models are externally published through the Low-Level Compliance utilities. Since the models are known, other agents can make use of the agent. Without model declaration, other agents are unaware of how to utilize the agent's services.

Complex Agents

The Resource, Model, And Wrap are required to properly define an agent. These base elements may be combined to form *Complex Agents*. Complex Agents allow for common functions to be centralized and for the sharing of agent utilities.

Multiple Models

Agents may contain multiple models. There may be a number of different operations that can be done with a resource depending on the information that is available or required. Those operations are made available through the use of different models. It is desired to use the model that can produce the required information with the best computational efficacy. An example of a multi-model agent would be a CATIA solids agent. Figure 7 shows a solid part that can be constructed from two different classes of operations. The first model involves solid construction based on Boolean entities, called **C**onstructive **S**olids **G**eometry (CSG) construction. The second model transforms a volume into the solid. Both models produce the same effective visualization result. For primitive shapes, the CSG method may be preferred. However, for complicated shapes, the volume transformation model would be required.

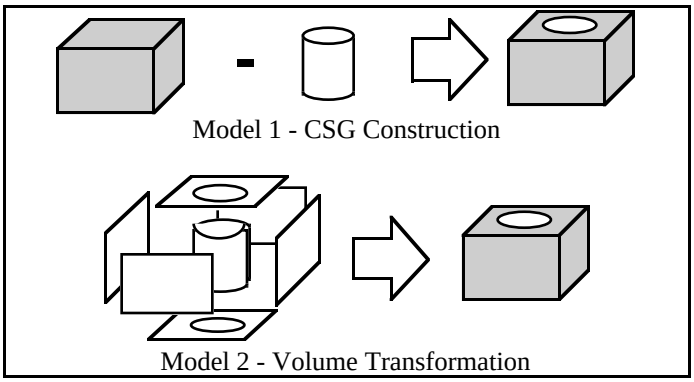

Figure 7. Multi-Model Example of a Solid

Multiple Resources

Agents may incorporate various resources to achieve their functionality. It is not required that the resources be implemented in the same manner since resource implementations are captured in the Implementation Model. An example of a multiple resource agent would be a database agent. The agent may include facilities for both relational and object-oriented storage capabilities.

Compound Agent

Agents may also include the use of other agents as resources. All services do not have to be provided by the agent nor are necessarily desired. By building agents from other agents, it is possible to independently optimize agent operations. An example of a compound agent is an aircraft design system. The system would incorporate the use of a geometric modeler agent, a database agent, and a sizing agent.

Benefits

The use of agents provides the computational facilities for the transformation of information into knowledge. Several computational issues are addressed and solved with the agent structure used here.

Flexibility

Agents provide the flexibility required to implement an integrated computing environment. Resources can be integrated without modifying original source code and execution scheme. The user may change code to take advantage of various agent utilities but is not required to do so. As discussed earlier, this independence allows for proprietary resources to be integrated across the design's life-cycle, see Figure 6. In addition, agents are platform independent (among UNIX operating systems). The same model and wrapping technologies can be applied to agents on a variety of platforms. The platform is usually dictated by that of the resource, which the user may not have the ability to modify.

Accountability

Social computing is playing an increasing role in the use of software.[19] Information generated by software products has been protected under a liability blanket. In design systems, this blanket is unacceptable and persons using this information need to be held responsible. Fortunately, agents are accountable. The agent not only produces new design knowledge, but the agent also adds additional design knowledge, including the who, what, when, and how. The model captures the additional information in the transformation that takes place within. Recall that the model has two components that are externally available, the Process Model and the Implementation Model.

Agent vs. Tool

Tools play an integral role in computing environments. Hence, the difference between the agent and the tool needs to be distinguished. The tool is an agent without a model. Therefore, information generated by the tool cannot be held accountable. Tools are valid design entities when used by design "experts" or in cases where no model exists. Tables 3-5 show a number of tools used to implement the IMAGE environment that have no model.

The tool is commonly referred to as the agent in other design systems.[2] Those systems employ new computing technologies to assist in the design process. Those "agents" are found to add only the communications and resource interpreter functions found in the IMAGE agent wrap to existing resources. Few, if any, formally associate models with resources. The focus on integration facilities rather than design support limits the usability of these tools for solving design problems.

Generic Agentization

A generic agentization scheme will enable designers to consistently integrate design resources into the IMAGE infrastructure. The following generic agentization scheme is under development.

Resource

Both proprietary and non-proprietary resources can be used. The only requirement on resources is that they do not conflict with the wrap and other resources. This requirement may present difficulty in the following situations:

- *Unit Assignments*. Hard-coded unit assignments (e.g. `OPEN(UNIT=11)`) will conflict when other resources use the same unit for IO purposes.
- *File Names*. Resources that use the same file names for IO will conflict when integrated.
- *Procedure Names*. Procedure references cannot be resolved for object code that has the same procedure calls as the wrap's shell.

The resource must be made secure in the open, integrated environment. One of the IMAGE tools that was proposed is a tool that will provide the capabilities for a secure environment, see Table 3.

Wrap

A tool that is successfully being used for agent wrapping in preliminary IMAGE implementations is the Tk/tcl utility package developed at U.C. Berkeley.[20] Tk/tcl is an interpretive windowing system.

The wrap has six components: the Communications Interface, the Protocol Filter, the Model Interpreter, the Resource Interpreter, a (Graphical) User Interface, and Low Level Compliance.

Communications Interface

The communications interface will use an event loop that polls the following communications channels:

- *User Input*. The user can interact through standard input/output of the agent.
- *X-Events*. X-events are processed and may include inter-interpreter events used by Tk/tcl.
- *Socket Interface*. The interface will have contain a run-time defined socket for communications. Parallel Virtual Machine (PVM) utilizes sockets to provide distributed processes that communicate via a sockets interface.[21] The Framework for Interdisciplinary Optimization (FIDO) enhances the interface by providing a utility layer (COMMLIB).[22]
- *Virtual File Transfer*. UNIX-domain, transparent file transfer can be used for synchronous data transfer across a virtual file domain.
- *Threads*. Threads can be used to re-direct data through standard program interface channels. These channels include stdio, stderr, etc. Threads also provide virtual process connection.[23, 24]

- *Hyper-Text Transfer Protocol*. HTTP allows for the transfer of multi-media type information across a transparent socket layer.
- *E-mail*. E-mail provides an asynchronous data transfer mode for platforms that do not have standard internet services.

Protocol Filter

Preliminary studies indicate that the Knowledge Query Manipulation Language (KQML) provides a suitable agent-based transfer format. The transfer format structure allows for designated and broadcast asynchronous data transfer. A suitable information encapsulation is still being investigated. As discussed earlier, the Knowledge Interchange Format (KIF) and natural language processing both offer advantages.

Model Interpreter

As with the protocol filter, a suitable information encapsulation method is being investigated.

Resource Interpreter

Resources are accessible by a generic resource interpreter. The interpreter is responsible for exchange of information with the resource in compliance with the process and implementation models. There are a number of ways that computer based resources can be accessed by the resource interpreter:

- *Interpreter Procedure*. The control flow of the resource may be altered so that the resource behaves like an interpreter procedure. This can only be done if source code is available (at least for the control routines) or a new control routine can be written (in this case only object code is needed).
- *tcl Script Procedure*. A new resource may be written in tcl and sourced as a run-time procedure.
- *Independent Process*. Resources not using stdio can be executed as their own processes. Information is exchanged with the process through data files or through UNIX file threads or the socket interface. In the case of threads and sockets, most resources will have to be altered to run in this manner, requiring source code. A WWW server is an example of an interface that is designed to run from a pre-defined port.
- *Threaded Process*. Resources using stdio can be threaded from the interpreter. Information can then be exchanged through the pipe that is created between them. Alternatively, stdio can be re-directed.

User-Interface

Tk/tcl provides a method for easily developing graphical user interfaces. However, it will be important to customize and standardize the GUIs across agent applications. One of the tools proposed for the IMAGE environment is an application defaults server that would be accessible to the various agents through the low-level compliance layer. In addition to the GUIs, a customizable tty-based interface will have to be developed for applications that do not have access to X libraries or an X display.

Low-Level Compliance

An agent independent, run-time configurable LLC layer is required for operation in an agent-based environment. The LLC layer will have the facilities to transparently (to the rest of the wrap) access the IMAGE environment tools.

Model

The model has two components: the implementation and process model. Again, a suitable language for describing models is being investigated.

Agent Demonstrations

A number of preliminary agents and tools were developed that demonstrate the technologies found in the generic agentization scheme. Tk/tcl, an interpretive windowing system, was used to implement the agents. The basic agents and tools include:

- *Geometric Modeler (CATIA™) - Agent.* A single function load was generated that provides full **CAT**IA **GEO**metry (CATGEO) access through the Tk/tcl interpretive shell.[25, 26] In addition, a volume transformation was defined that creates solid part representations based on parametric definitions. The transformation follows the model shown Figure 6. Other agents may utilize the Geometric Modeler as an agent by using the transformation model or as a tool by circumventing the model and using the CATGEO resources directly. This agent demonstrates the ability to make a fully-defined agent using a commercial resource. Therefore, no source code is required.
- *Database (ORACLE™) - Tool.* The Pro-C programming library provided by ORACLE was utilized to produce a single Tk/tcl function that interprets SQL commands.[27] SQL queries are dynamically managed by the load and processed in the ORACLE database.
- *Communications (PVM) - Tool*. The **P**arallel **V**irtual **M**achine (PVM) functions were added to the event-loop of Tk/tcl as well as a Tk/tcl procedure was added for each PVM function.[21] Therefore, a reliable, networked channel is available for agent communications.
- *Expert System (CLIPS) - Tool*. The **C** **L**anguage **I**ntegrated **P**roduction **S**ystem (CLIPS) was added as a procedure to the Tk/tcl interpreter.[28] Therefore, a single rule base is available to external agents and tools.

Compound agents and tools were built from these basic interfaces. These services include:

- *Wing Visualization (CATIA) - Agent*. A **P**assively **C**ontrolled **L**ifting **S**urface (PCLS) definition is generated within this agent, see Figure 8. In turn, the volume transformation model found in the Geometric Modeler Agent is used to render the PCLS in CATIA real-time.
- *World Wide Web Interface (Netscape, CATIA) - Tool*. A World Wide Web interface was developed using html that allows basic geometry to be rendered in CATIA though the Geometric Modeler Agent described above, see Figure 9. The geometry is rendered in CATIA running in interactive mode on a remote machine and graphics are returned real-time. This interface demonstrates the collaboration of proprietary and other resources across a network.

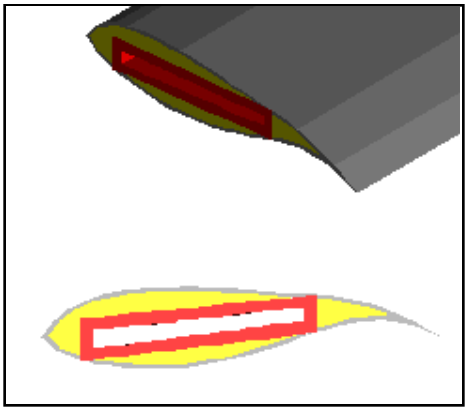

Figure 8. PCLS in CATIA

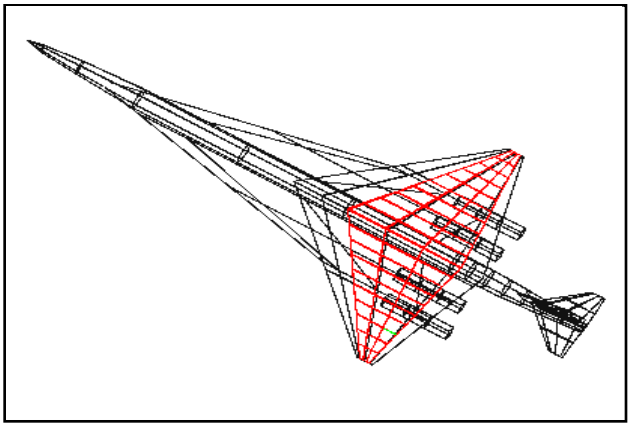

Figure 10. ACSYNT Defined Model in CATIA

Figure 9. WWW Interface to CATIA

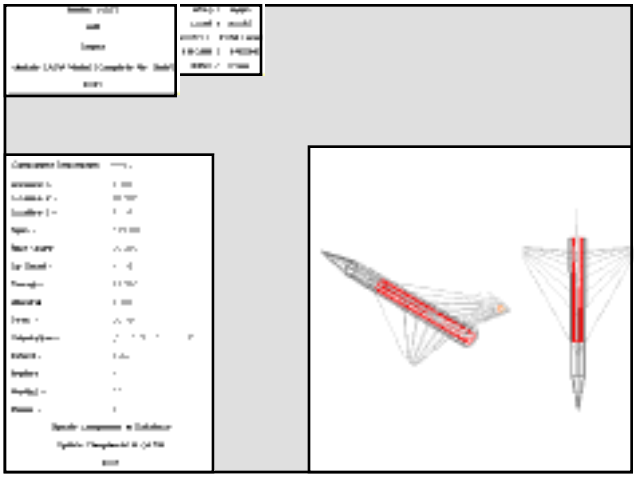

Figure 11. Aircraft Design System

- *Aircraft Visualization (CATIA, ACSYNT) - Tool*. This tool automatically renders ACSYNT (an aircraft convergence code) defined geometry in CATIA as a wireframe model, see Figure 10. The geometry is obtained from ACSYNT as a HERMITE file. The file is translated run-time and the wireframe objects are created in CATIA. This tool was written in less than an hour and shows the ease of developing inter-tool file processing.
- *Aircraft Design System (CATIA, ORACLE) - Tool*. The aircraft design system uses Tk/tcl defined user interfaces to build parametric descriptions of aircraft components, see Figure 11. The descriptions are stored in the ORACLE tool described above and the components are rendered in CATIA real-time. This system demonstrates the ability for multiple proprietary resources to be used in an agent-based framework.

Future Directions

Having demonstrated the technologies required to implement agents, research in the development of the IMAGE computing infrastructure is continuing. The following areas are currently being addressed:

- *Harness computing technologies that utilize existing engineering practices*. Many of the new computing technologies offer significant advance in computing power but have seen limited application in engineering programs. The new technologies are not backward compatible with existing code and they require constructs not found in traditional programming languages.
- *Formalize a generic agentization scheme*. A generic agentization scheme will enable designers to consistently integrate design resources into the IMAGE infrastructure. Three components have been identified for the agent: the Resource, the Model, and the Wrap.
- *Develop the agents and tools required to implement the IMAGE infrastructure*. Agents and tools required to implement the IMAGE computing infrastructure are outlined in Tables 1-5. The agents/tools will be developed as the research progresses and demonstration problems mature.

When in place, the new architecture, supported by the computing infrastructure, will result in a system that facilitates designing from a decision-based perspective. Thus, a designer will have the capability to producing better designs while expending fewer resources. A designer will have more

knowledge, that is complete and structured, available during decision-making processes.

Acknowledgments

Funding for this paper is provided under the Graduate Student Researchers Program (NGT-51250) directed by NASA Langley High Performance Computing and Communications Program and the NASA MDO Program (NAG-1-1564).

References

[1] "ACSYNT Overview and Installation Manual," ACSYNT Institute, Virginia Polytechnic Institute and State University, May 1992.

[2] Cutkosky, M. R., R. S. Engelmore, R. E. Fikes, M. R. Genesereth, T. R. Gruber, W. S. Mark, J. M. Tenenbaum and J. C. Weber, "PACT: An Experiment in Integrating Concurrent Engineering Systems," IEEE Computer, pp. 28-37, January, 1993.

[3] Dovi, A. R., G. A. Wrenn, J.-F. M. Barthelemy, P. G. Coen and L. E. Hall, "Multidisciplinary Design Integration System for a Supersonic Transport Aircraft," Fourth AIAA / USAF / NASA / OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, OH, September 21-23, 1992. AIAA-92-4841.

[4] Hughes, D., "Generic Command Center Speeds Systems Design," Aviation Week & Space Technology, pp. 52-53, March 8, 1993.

[5] Jones, K. H., D. P. Randall and C. K. Cronin, "Information Management for a Large Multidisciplinary Project," Fourth AIAA / USAF / NASA / OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, OH, September 21-23, 1992. AIAA-92-4720.

[6] Gage, P. and I. Kroo, "Development of the Quasi-Procedural Method for Use in Aircraft Configuration Optimization," Fourth AIAA / USAF / NASA / OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, OH, September 21-23, 1992. AIAA-92-4693.

[7] Kroo, I. and M. Takai, "A Quasi-Procedural, Knowledge-Based System for Aircraft Design," AIAA / AHS / ASEE Aircraft Design, Systems and Operations Meeting, Atlanta, GA, September 7-9, 1988. AIAA-88-4428.

[8] McCullers, L. A., "FLight OPtimization System, User's Guide, Version 5.41," NASA Langley Research Center, December, 1993.

[9] F. Mistree, W. F. Smith and B. A. Bras, *A Decision-Based Approach to Concurrent Engineering*, Ed. Paresai, H. R. and W. Sullivan, Handbook of Concurrent Engineering, Chapman & Hall, New York, 1993. (pp. 127-158).

[10] Bras, B. A. and F. Mistree, "Desiging Design Processes in Decision-Based Concurrent Engineering," SAE Transactions Journal of Materials & Manufacturing, pp. 451-458, Warrendale, PA, SAE International, 1991.

[11] Mistree, F., W. F. Smith, B. A. Bras, J. K. Allen and D. Muster, "Decision-Based Design: A Contemporary Paradigm for Ship Design," Transactions, Society of Naval Architects and Marine Engineers, Jersey City, New Jersey, pp. 565-597, 1990.

[12] B. A. Bras, W. F. Smith and F. Mistree, *The Development of a Design Guidance System for the Early Stages of Design*, Ed. Oortmerssen, G. V., CFD and CAD in Ship Design, Elsevier Science Publishers B.V., Wageningen, The Netherlands, (pp. 221-231).

[13] F. Mistree, O. F. Hughes and B. A. Bras, *The Compromise Decision Support Problem and the Adaptive Linear Programming Algorithm*, Ed. Kamat, M. P., Structural Optimization: Status and Promise, Washington, DC, (pp. 247-286), AIAA.

[14] Hale, M. A., "IMAGE: An Intelligent Multidisciplinary Aircraft Generation Environment," Masters Program Special Project, Georgia Institute of Technology, School of Aerospace Engineering, September, 1992.

[15] Stephens, E., "LEGEND: Laboratory Environment for the Generation, Evaluation, and Navigation of Design," Doctoral Dissertation, Georgia Institute of Technology, School of Aerospace Engineering, September 1993.

[16] Finin, T., J. Weber, G. Wiederhold, M. Genesereth, R. Frtzon, D. McKay, J. McGuire, R. Pelavin, S. Shapiro and C. Beck, "Specification of the KQML Agent-Communication Language," The DARPA Knowledge Sharing Initiative External Interfaces Working Group, February, 1994.

[17] T. R. Gruber and G. R. Olsen, *An Ontology for Engineering Mathematics*, Ed. Doyle, J., P. Torasso and E. Sandewall, Fourth International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann, Bonn, Germany, 1994.

[18] Genesereth, M. R. and N. P. Singh, "A Knowledge Sharing Approach to Software Interoperation," January, 1994.

[19] Nissenbaum, H., "Computing and Accountability," Communications of the ACM, pp. 72-81, January, 1994.

[20] Ousterholt, J. K., An Introduction to Tcl and Tk, Addison-Wesly Publishing Company, Inc., Reading, MA, 1993.

[21] "Parallel Virtual Machine User's Manual," Version 2.4, June, 1993.

[22] Eidson, T. M., "FIDO Project: Communication Library Usage Guide," NASA Langley Research Center, July, 1994.

[23] Chapman, B., P. Mehrotra, J. V. Rosendale and H. Zima, "A Software Architecture for Multidisciplinary Applications: Integrating Task and Data Parallelism," Institute for Computer Applications in Science and Engineering, March 1994. NASA CR-194896, ICASE 94-18.

[24] Mehrotra, P. and M. Haines, "An Overview of the OPUS Language and Runtime System," Institute for Computer Applications in Science and Engineering, May 1994. NASA CR-194921, ICASE 94-39.

[25] "CATIA Geometry Interface Reference Manuals, Version 3 Release 2.5," Dassault Systemes, December, 1990.

[26] Hale, M. A. and J. I. Craig, "Preliminary Development of Agent Technologies for a Design Integration Framework," AIAA / NASA / USAF / ISSMO Symposium on Multidisciplinary Analysis and Optimization, Panama City, Florida, September 7-9, 1994. AIAA-94-4297.

[27] "ORACLE Pro*C User's Guide, Version 1.1," ORACLE Corporation, April, 1987.

[28] "C Language Integrated Production System (CLIPS) User's Manual, Version 6.0," NASA Johnson Space Center.